Design Example
(Class 11.2 – 4/4/2013)

CSE 2441 – Introduction to Digital Logic
Spring 2013
Instructor – Bill Carroll, Professor of CSE
Today’s Topic -- Design Example

Design a digital lock/display controller.

\[
\begin{array}{c|c|c|c|c}
\text{7-Seg} & \text{7-Seg} & \text{7-Seg} & \text{7-Seg} \\
\hline
\text{X1} & \text{X0} & \text{Enter} & \text{Reset/Lock} \\
\hline
\end{array}
\]

**Digital Lock/Display Controller**

- **X1** and **X0** are used to input the binary code sequences for unlocking the lock.
- **Enter** is used to load the current input values.
- **Reset/Lock** is used to restart the code entry process or to lock the lock after it’s opened.
- The lock will be opened (**Open/Lock = 1**) if and only if code sequence **X1X0: 00-10-01** is entered. Access will be denied (**Deny = 1**) when any other length-three, 2-bit binary sequence is entered.
- **DenyOpen/Lock = d0** indicates that the lock is locked. **DenyOpen/Lock = 11** is undefined.
- The controller must stay in the denied or opened state until reset or locked.
- The state of the lock (**locked, open**) must be shown on the display. The display should show **deny** after an invalid sequence has been entered.
Design Specifications

• Design a finite state machine controller that meets the following criteria.
  – Satisfies the above I/O definitions.
  – Uses the minimum number of flip flops.
  – Uses the minimum number of combinational logic elements (gates, decoders, multiplexers, etc.).
  – Does not produce any unwanted or undefined outputs.

• Can be prototyped on an Altera DE-1 board. Remember that the seven-segment displays on the DE-1 are active-low (common anode) devices.
Hierarchical Design

- Top-down design
- Divide-and-conquer
Finite State Machine – State Diagram
(Mealy or Moore?)

<table>
<thead>
<tr>
<th>PS</th>
<th>x1x0</th>
<th>z1z0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
</tbody>
</table>

Nomenclature:
S/z1z0
z1 = Deny, z0 = Open/Lock

Next States
Equivalent States/State Assignment

• Using partitioning
  
  \[ P_0 = (ABCD\overline{EFG}) \]
  
  \[ P_1 = (ABCDE)(\overline{D})(\overline{G}) \]
  
  \[ P_2 = (AE)(B)(\overline{C})(F)(\overline{D})(\overline{G}) \]
  
  \[ P_3 = (A)(E)(B)(C)(F)(\overline{D})(\overline{G}) \]

• No equivalent states

• State assignment
  
  \[ A \rightarrow 000 \]
  
  \[ B \rightarrow 001 \]
  
  \[ C \rightarrow 011 \]
  
  \[ D \rightarrow 010 \]
  
  \[ E \rightarrow 100 \]
  
  \[ F \rightarrow 101 \]
  
  \[ G \rightarrow 111 \]
## Produce a Transition Table

<table>
<thead>
<tr>
<th>y2y1y0</th>
<th>x1x0</th>
<th>z1z0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>011</td>
<td>111</td>
<td>010</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>101</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>ddd</td>
<td>ddd</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Control-Output Logic Design

\[ z_1 = y_2y_1 \]

\[ z_0 = y_1y_0' \]
Finite State Machine – Flip-Flop Selection (D or JK?)

- **D flip-flop input equations**
  - \( Y_2 = x_0y_1' + x_1y_1'y_0' + x_1'x_0'y_0 + x_1y_1y_0 + y_2 \)
  - \( Y_1 = y_1 + y_2y_0 + x_1x_0'y_0 \)
  - \( Y_0 = y_1'y_0 + x_1y_0 + x_0'y_0 + x_1'x_0'y_1' + y_2 \)

- **JK flip-flop input equations**
  - \( J_2 = x_0y_1' + x_1y_1'y_0' + x_1'x_0'y_0 + x_1y_1y_0 \)
  - \( K_2 = y_2' \)
  - \( J_1 = y_2y_0 + x_1x_0'y_0 \)
  - \( K_1 = y_1' \)
  - \( J_0 = x_1'x_0'y_1' + y_2 \)
  - \( K_0 = x_1'x_0y_2'y_1 \)
Reset/Lock and Enter Inputs

- Use Clear input of FF for R/L
- Use Clock input for Enter
Hierarchical Design

- Top-down design
- Divide-and-conquer
# Display Decoder Design

<table>
<thead>
<tr>
<th>Deny</th>
<th>Open</th>
<th>7-seg A</th>
<th>7-seg B</th>
<th>7-seg C</th>
<th>7-seg D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>L</td>
<td>o</td>
<td>C</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>o</td>
<td>P</td>
<td>E</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>d</td>
<td>E</td>
<td>n</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## 7-Seg A Decoder

<table>
<thead>
<tr>
<th>Deny</th>
<th>Open</th>
<th>Char</th>
<th>Seg-a</th>
<th>Seg-b</th>
<th>Seg-c</th>
<th>Seg-d</th>
<th>Seg-e</th>
<th>Seg-f</th>
<th>Seg-g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

\[
a' = 0
b' = \text{Deny}
c' = \text{Deny} + \text{Open}
d' = 1
e' = 1
f' = (\text{Deny})'(\text{Open})'
g' = \text{Deny} + \text{Open}
\]
Final Design for DE-1 Implementation